DATA SECURITY FOR DISTRIBUTED FILE SYSTEMS

Inventors

Lance W. Russell
1290 Versailles Dr.
Hollister, CA 95023

Lu Xu
1252 Oak Knoll Dr.
San Jose, CA 95129

Assignee

Hewlett Packard Company

DATA SECURITY FOR DISTRIBUTED FILE SYSTEMS

FIELD OF THE INVENTION

The present invention generally relates to data security for distributed file systems, and more particularly to dynamically created objects for enforcing security in distributed file systems.

BACKGROUND

The growth of the Internet is driving the need for additional data storage capacity. While the need for data storage devices is readily met with additional storage devices, performance is often hindered by the manner in which access is provided to the data.

Many computer networks provide access to data through storage controllers, each of which controls access to one or more storage devices. A storage controller may create a bottleneck that limits data throughput if many client applications seek access to data controlled by the same storage controller, or each if a few client applications move a large quantity of data through the same storage controller.

A current solution to the problem is the used of network-attached secure devices (NASD). The NASD architecture allows client nodes to directly access specific storage objects by supporting a cache capability. The cache allows file data and some control information to be transmitted to a client node only once via the network, and there is no centralized storage controller to create a bottleneck. While this addresses the bottleneck issue, data security must now be addressed at the storage device level.

A common NASD architecture provides a NASD interface and file system where files and directories are stored in NASD objects. Each file and directory occupies a dedicated NASD object. Security is based on cryptographic attributes of the objects, and the objects are static. That is, a file object is created when the file is created and exists as long as the file exists. Furthermore, objects are inherently supported at the device level and each is therefore limited by the device with which the object is associated.

Since each file has an associated object, the client applications that access the files must be programmed to handle the objects. Thus, a major software upgrade may be required to implement a NASD-based system with existing client applications. In

addition, enforcing security at the file level may create excess system overhead through encryption and key management.

A system and method that address the aforementioned problems, as well as other related problems, are therefore desirable.

SUMMARY OF THE INVENTION

In various embodiments, the invention provides data security in a distributed file system. A distributed file system interface is coupled to the one or more client applications, and a storage server and a meta-data server are coupled to the distributed file system interface. The meta-data server receives open-file requests from the distributed file system interface and in response creates a security object. The meta-data server also generates an partial encryption key and stores the partial encryption key in the security object. The block storage server completes the encryption key, and the meta-data server encrypts the list of blocks that are in the file and stores the encrypted block list in the security object. The security object is then returned to the distributed file interface and used in subsequent file access requests.

It will be appreciated that various other embodiments are set forth in the Detailed Description and Claims which follow.

BRIEF DESCRIPTION OF THE DRAWINGS

Various aspects and advantages of the invention will become apparent upon review of the following detailed description and upon reference to the drawings in which:

FIG. 1 is a functional block diagram that illustrates the data flow between a client application, meta-data server, and block storage server in accordance with one embodiment of the invention;

FIG. 2 is a flowchart of a process performed by a meta-data server in opening a file;

FIG. 3 is a flowchart of a process performed by a storage server in servicing an open file request;

FIG. 4 is a flowchart of a process performed by a meta-data server in closing a file;

FIG. 5 is a flowchart of a process performed by a storage server in closing a file;

FIG. 6 is a flowchart of a process performed by a distributed file system interface in processing a file access request; and

FIG. 7 is a flowchart of a process performed by a storage server in processing a file access request.

DETAILED DESCRIPTION

FIG. 1 is a functional block diagram that illustrates the data flow between a client application, meta-data server, and block storage server in accordance with one embodiment of the invention. Data security is provided by meta-data server 102, distributed file system interfaces 104a and 104b, and block storage server 106. By implementing the functions of meta-data server 102 on a separate system from storage server 106, the bottlenecks associated with a storage controller implementation are avoided. In addition, since security is handled by the distributed file system interface 104a, 104b, in conjunction with the meta-data server 102 and the storage server 106, the client application does not need to be specially programmed for security considerations and associated system overhead on the client system is avoided.

Meta-data server 102 services file access requests for one or more distributed file system interfaces 104a, 104b. Each of the distributed file system interfaces services one or more client applications. For example, distributed file system interface 104a services client application 108a, and distributed file system interface 104b services client application 108b. The file data in the distributed file system is stored by one or more block storage servers, for example, block storage server 106.

In one embodiment, meta-data server is implemented on a data processing system that is coupled to the distributed file system interfaces 104a, 104b and to the storage server 106 by a network, such as a LAN or WAN, for example. Similarly, each distributed file system interface is implemented on a separate data processing system. The distributed file system interface can be implemented, for example, as shown in the patent/application entitled, "EXTENDING A STANDARD-BASED REMOTE FILE ACCESS PROTOCOL AND MAINTAINING COMPATIBILITY WITH A STANDARD PROTOCOL STACK" having patent/application number 09/774,841, filed on January 31, 2001 by Karamanolis et al., the contents of which are

incorporated herein by reference. Block storage server 106 can be implemented using conventional block-based storage server systems. Those skilled in the art will appreciate that various alternative data processing arrangements are adaptable to provide the capabilities of the present invention.

Meta-data server 102 manages meta-data and security for files under its control. Meta-data includes: file I-nodes, block bitmaps, ownership, access times, system parameters, and other information that describes the files in the system. The i-node includes a list of blocks in the file, which the meta-data server 102 passes to the block storage server 106.

Security is managed on two levels. On the first level, read/write permission to files is managed using permission codes that are associated with different client applications. For example, a first client application may have only read permission to a file, while a second client application has read and write permissions to the file. On the second level, the meta-data server manages secure transmission of file data over the network that connects the storage server 106 and the distributed file system interfaces 104a, 104b. The secure transmission of data is accomplished using an encryption key that is transmitted in a security object between the meta-data server 102, distributed file system interfaces 104a, 104b, and storage server 106.

The permission codes for a file are maintained by the meta-data server 102 and configured when a file is created or when a client application needs access to a file. The manner in which the meta-data server 102 maintains the permission codes with storage server 106 and the management of encryption keys for encrypting file data transmitted over a network alleviates the storage controller bottleneck of prior systems while providing for secure transmission of data.

FIG. 1 illustrates an example interaction between the client applications 108a and 108b and the distributed file system in opening files named "*foo*" and "*bar*." Client application 108a uses distributed file system interface 104a to open *foo*. The open file request is transmitted to the meta-data server 102 as shown by line 122. The meta-data server creates a security object (if one does not already exist) for the protection domain of which the requesting client application is a member. In this example, client application 108a, and client application 108b are in the same protection domain and have access to data on storage server 106 using the same encryption key.

In response to the open file request, the meta-data server 102 first generates an encryption key. Then the security object, along with the open file request, is transmitted to the storage server 106 as illustrated by ellipse 124. The security object includes a file identifier, encryption key, and a permission code that is associated with the client application. The security key is passed between components because the keys are created collaboratively (for security), and the components will use them to decrypt the information. Block storage server 106 receives the security object and generates a list of blocks in the referenced file. The block list generally includes enough information for the block server to locate the data in subsequent requests from the client application, and the specific information is implementation dependent. The block list is then encrypted using the encryption key in the security object and stored in the security object, and the updated security object is returned to the meta-data server 102, as shown by ellipse 126. The meta-data server 102 then returns the security object to the distributed file system interface 104a as shown by ellipse 128. The distributed file system interface 104a then returns a status code to the client application 108a. Note that the security object remains with the distributed file system interface 104a, and the client application 108a does not have access to nor need the encryption key.

When client application 104a issues a read request for *foo*, the distributed file system interface forwards the read request, along with the encrypted block list and encryption key from the security object, to the storage server 106, as illustrated by line 130. The storage server uses the encryption key to decrypt the block list. The storage server 106 responds by retrieving and encrypting the requested data and then returning the encrypted data as shown by line 134.

Independent of client application 108a, client application 108b issues a request to open *bar*. The distributed file system interface 104b responds by transmitting an open file request to the meta-data server 102 as shown by line 136. Since client applications 108a and 108b are in the same security domain, the meta-data server adds the file identifier for *bar* to the security object and transmits the security object to the storage server 106 as shown by ellipse 138. Since the client applications 108a and 108b are in the same security domain, the same encryption methodology is used for both *foo* and *bar*. However, a new encryption key is created each time a file is opened. A dedicated security object is established for each security domain. Thus, each distributed file system interface is configured to service client applications in different

security domains by using the security objects as they are passed by the meta-data server.

The storage server 106 responds to the request to open *bar* by encrypting the *bar* block list and adding the encrypted block list to the security object. The updated security object is then returned to the meta-data server as shown by ellipse 140. The meta-data server 102 returns the security object to the distributed file system interface 104b as shown by ellipse 142.

FIG. 2 is a flowchart of a process performed by a meta-data server in opening a file in accordance with one embodiment of the invention. The process assumes that the file to be opened was already created and populated with data. In addition, read/write permissions for different client applications are associated with the files by the meta-data server according to application requirements.

At step 202, if the open file request is the first request by a client application in a particular security domain, the meta-data server generates a partial encryption key for use with the security object. Those skilled in the art will recognize that various encryption techniques may be suitable, depending on implementation requirements. Depending on implementation requirements, the same technique could be used in all the security domains to establish the encryption keys. Alternatively, different techniques could be used in different security domains. At step 204, the security object is transmitted to the storage server, wherein the security object includes the file identifier of the requested file, and the partial encryption key.

The storage server returns information necessary for the meta-data server to complete the encryption key, and at step 206, the meta-data server receives the information from the storage server. At step 208, the block list that is maintained by the meta-data server is encrypted and added to the security object, and at step 210 the security object is returned to the distributed file system interface. The meta-data server saves copies of all security objects that are associated with the blocks of open files, along with associations with the client systems with which the security objects are associated. While not shown, it should be understood that the distributed file system interface returns a status code to the requesting client application and saves the security object for subsequent file access requests. Thus, the client application need not be programmed to handle secure data transmissions.

The block list in the security object is updated by the meta-data server. When a file is truncated, for example, the meta-data server revokes all open accesses for the impacted blocks before the write is allowed to occur. Subsequent reads of these blocks proceed with new objects. Those skilled in the art will appreciate that the meta-data server is involved in file operations that result in changes to the block list. For example, in processing a write request in one implementation, the distributed file system interface submits a write request to the metadata server. In response, the meta-data server locks the section of the file being written and makes any necessary changes to the block list.

FIG. 3 is a flowchart of a process performed by the storage server in servicing an open file request in accordance with one embodiment of the invention. At step 302, the storage server receives an open file request and the security object from the meta-data server. The open file request includes the file identifier of the file to open, and the security object includes the partial encryption key for the security domain in which the file is opened.

At step 304, the storage server completes the encryption key and stores it in the security object. At step 306, the encryption key is stored for later use. When the storage server receives a file access request along with an accompanying security object from a client, the saved encryption key is used to decrypt the block list from the security object. If the block list is successfully decrypted, the client is provided with the requested access. At step 308, the updated security object is returned to the meta-data server.

FIG. 4 is a flowchart of a process performed by the meta-data server in closing a file. At step 352, the meta-data server receives a close file request from a distributed file system interface. The distributed file system interface also transmits the security object to the meta-data server along with the close file request.

At step 354, the close file request and security object are transmitted to the storage server. The meta-data server then waits for a response from the storage server before continuing the process. At step 356, the meta-data server receives the updated security object from the storage server and removes the block list of the requested file from the security object.

At step 358, the meta-data server checks whether there are any remaining block lists in the security object. If not, the security object is deleted, and a status code is

returned to the distribute file system interface. Otherwise, the security object is returned to the distributed file system interface.

FIG. 5 is a flowchart of a process performed by a block storage server in closing a file. At step 402, the block storage server receives a close file request from the meta-data server along with the security object for the security domain to which the requesting client application belongs. At step 404, the storage server invalidates the security object within the storage server so that any further requests with the invalidated object will be refused. At step 406, the security object is returned to the meta-data server.

FIG. 6 is a flowchart of a process performed by the client distributed file system interface in processing a file access request. At step 452, the request is received from the client application, and at step 454 a file access request is generated and sent to the storage server having the file data. The request to the storage server includes the encryption key from the security object for the client.

At step 456, the distributed file system interface processes the interaction with the storage server in accordance with the type of file access request. For example, if the request is a read, the distributed file system interface awaits data from the storage server. When the data are received, the distributed file system interface decrypts the data and returns the data to the client application.

For a write request, the distributed file system interface encrypts the data provided by the client application and sends the encrypted data to the storage server. The distributed file system interface also returns status codes from the storage server to the client application.

FIG. 7 is a flowchart of a process performed by the storage server in processing a file access request. At step 502, the storage server receives from a distributed file system interface a file access request. The file access request includes a file identifier, an operation code, an encryption key, and an encrypted list of the blocks of data to manipulate.

At step 503, the encrypted list of blocks is decrypted using the key from the request. Decision step 504 tests whether the decryption was successful. If the decryption was successful, the block storage server assumes that the client application has the necessary access rights. Otherwise, the process is directed to step 506 where a

status code is returned to the distributed file system interface to indicate that access was denied.

If the client application has the necessary access rights, decision step 504 directs the process to step 508. At step 508, the operation specified by the operation code is performed. If the distributed file system interface is reading data, the data read are returned to the distributed file system interface. If the distributed file system interface is writing data, the encrypted data received from the distributed file system interface is written to the storage device. At step 510, the response/data are returned to the distributed file system interface.

Embodiments of the present invention, beyond those described herein, will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein. It is intended that the specification and illustrated embodiments be considered as examples only, with a true scope and spirit of the invention being indicated by the following claims.